1. Multicast

There are some scenarios we need the Multicast. Such as that several users want to receive the packets from the server. It makes the server can send packets to the clients who want to receive the packets and unlike the Unicast only can send the packets to one client.

1. Multicast address

All the clients should associate with a multicast address that the server will send to.

MULTICAST Address 224.0.0.0 - 239.255.255.255

Local multiple address 224.0.0.0～224.0.0.255 Local area network

Internet multiple address 224.0.1.0～238.255.255.255 Internet network

Private multiple address 239.0.0.0～239.255.255.255 Private network

1. Multicast programming

Opts of Multicast

|  |  |
| --- | --- |
| **getsockopt()/setsockopt()** |  |
| IP\_ADD\_MEMBERSHIP | Add local interface to the Multiple address group |
| IP\_DROP\_MEMBERSHIP | Drop local interface the Multiple address group |
| IP\_MULTICAST\_IF | Set the interface to send the packet |
| IP\_MULTICAST\_LOOP | Set loop multicast packet |

**Receiver:**

1. **IP\_ADD\_MEMBERSHIP**: Add the interface of the LOCAL\_ADDR to the MCAST\_ADDR group

#define MCAST\_ADDR "224.0.0.88"

struct ip\_mreq mreq;

mreq.imr\_multiaddr.s\_addr = inet\_addr(MCAST\_ADDR);

mreq.imr\_interface.s\_addr = inet\_addr(LOCAL\_ADDR);// add ip interface to mcast\_addr group

err = setsockopt(s, IPPROTO\_IP, IP\_ADD\_MEMBERSHIP, &mreq, sizeof(mreq));

if (err < 0)

{

perror("setsockopt():IP\_ADD\_MEMBERSHIP");

return -4;

}

1. **IP\_DROP\_MEMBERSHIP**

err = setsockopt(s, IPPROTO\_IP, IP\_DROP\_MEMBERSHIP, &mreq, sizeof(mreq));

if (err < 0)

{

perror("setsockopt():IP\_DROP\_MEMBERSHIP");

return -4;

}

**Sender:**

1. **IP\_MULTICAST\_IF**: if the socket do bind to any local ip, then the os will fill the source ipaddr as the the following one.

struct in\_addr localInterface;

localInterface.s\_addr = inet\_addr(LOCAL\_ADDR);

if (setsockopt(s, IPPROTO\_IP, IP\_MULTICAST\_IF, (char \*) &localInterface, sizeof(localInterface)) <0)

{

perror("setsockopt():IP\_MULTICAST\_IF ");

return -1;

}

1. **IP\_MULTICAST\_LOOP**: It makes the multicast packets send back to host too. If there are clients on the same host or the sender is in the same Muticast address group. All these clients or itself can also receive the packets.

int loop = 1;

err = setsockopt(s, IPPROTO\_IP, IP\_MULTICAST\_LOOP, &loop, sizeof(loop));

if(err < 0)

{

perror("setsockopt():IP\_MULTICAST\_LOOP");

return -1;

}

1. Multicast example

Tips:

1. Receive local bind address should be INADDR\_ANY
2. The port of receiver bind must be equal to the one the sender sendto
3. If the sender and receiver on the same host the IP\_MULTICAST\_LOOP should enable

**Sender:**

#define MCAST\_ADDR "224.0.0.88"

#define MCAST\_DATA "MULTICAST TEST DATA"

#define MCAST\_INTERVAL 2

int main(int argc, char\*argv[])

{

int s, n, err;

struct sockaddr\_in mcast\_addr;

struct in\_addr localInterface;

if (argc != 3)

{

printf("%s local\_ip remote\_port\n", argv[0]);

return -1;

}

s = socket(AF\_INET, SOCK\_DGRAM, 0);

if (s == -1)

{

perror("socket()");

return -1;

}

memset(&mcast\_addr, 0, sizeof(mcast\_addr));

mcast\_addr.sin\_family = AF\_INET;

mcast\_addr.sin\_addr.s\_addr = inet\_addr(MCAST\_ADDR);

mcast\_addr.sin\_port = htons(atoi(argv[2]));

localInterface.s\_addr = inet\_addr(argv[1]);

if (setsockopt(s, IPPROTO\_IP, IP\_MULTICAST\_IF, (char \*) &localInterface, sizeof(localInterface)) < 0)

{

perror("setsockopt");

return -1;

}

int loop = 1;

err = setsockopt(s, IPPROTO\_IP, IP\_MULTICAST\_LOOP, &loop, sizeof(loop));

if(err < 0)

{

perror("setsockopt():IP\_MULTICAST\_LOOP");

return -3;

}

while(1)

{

n = sendto(s, MCAST\_DATA, sizeof(MCAST\_DATA), 0, (struct sockaddr\*)&mcast\_addr, sizeof(mcast\_addr)) ;

if( n < 0)

{

perror("sendto()");

return -2;

}

sleep(MCAST\_INTERVAL);

}

return 0;

}

**Receiver:**

#include <stdio.h>

#include <sys/prctl.h>

#include <assert.h>

#include <sys/types.h>

#include <sys/stat.h>

#include <fcntl.h>

#include <stdlib.h>

#include <unistd.h>

#include <sys/syscall.h>

#include <sys/wait.h>

#include <sched.h>

#include <sys/socket.h>

#include <string.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#include <sys/epoll.h>

#define MCAST\_ADDR "224.0.0.88"

#define MCAST\_INTERVAL 5

#define BUFF\_SIZE 256

static int efd;

/\* change socket to nonblocking mode \*/

int make\_socket\_nonblocking(int sfd)

{

int flags, s;

flags = fcntl(sfd, F\_GETFL, 0);

if (flags == -1)

{

perror("fcntl F\_GETFL");

return -1;

}

flags |= O\_NONBLOCK;

s = fcntl(sfd, F\_SETFL, flags);

if (s == -1)

{

perror("fcntl F\_SETFL");

return -1;

}

return 0;

}

int epoll\_init(unsigned int max\_events)

{

efd = epoll\_create(max\_events);

if (efd == -1)

return -1;

return 0;

}

int main(int argc, char\*argv[])

{

int s;

struct sockaddr\_in local\_addr;

int err = -1;

if (argc != 3)

{

printf("%s local\_ip port\n", argv[0]);

return -1;

}

s = socket(AF\_INET, SOCK\_DGRAM, 0);

if (s == -1)

{

perror("socket()");

return -1;

}

/\* Enable SO\_REUSEADDR can bind several clients on the same ip an port \*/

int reuse = 1;

if (setsockopt (s, SOL\_SOCKET, SO\_REUSEADDR, (char \*) &reuse, sizeof(reuse)) < 0)

{

perror("Setting SO\_REUSEADDR error");

close(s);

return -1;

}

memset(&local\_addr, 0, sizeof(local\_addr));

local\_addr.sin\_family = AF\_INET;

local\_addr.sin\_addr.s\_addr = htonl(INADDR\_ANY);//must be this

local\_addr.sin\_port = htons(atoi(argv[2]));

err = bind(s, (struct sockaddr\*)&local\_addr, sizeof(local\_addr)) ;

if(err < 0)

{

perror("bind()");

return -2;

}

err = make\_socket\_nonblocking(s);

if (err != 0)

{

perror("nonblocking()");

close(s);

return -1;

}

struct ip\_mreq mreq;

mreq.imr\_multiaddr.s\_addr = inet\_addr(MCAST\_ADDR);

mreq.imr\_interface.s\_addr = inet\_addr(argv[1]);// add ip interface to mcast\_addr group

err = setsockopt(s, IPPROTO\_IP, IP\_ADD\_MEMBERSHIP, &mreq, sizeof(mreq));

if (err < 0)

{

perror("setsockopt():IP\_ADD\_MEMBERSHIP");

return -4;

}

err = epoll\_init(10);

if (err < 0)

{

perror("epoll\_init");

return -4;

}

struct epoll\_event ev;

ev.events = EPOLLIN;

ev.data.fd = s;

err = epoll\_ctl(efd, EPOLL\_CTL\_ADD, s, &ev);

if (err < 0)

{

perror("epoll\_ctl");

return -1;

}

int times = 0;

char buff[BUFF\_SIZE];

int n = 0;

struct epoll\_event events[10];

int i, ne;

while(1)

{

ne = epoll\_wait(efd, events, 10, -1);

if (ne != 0)

printf("waiting ... got %d events\n", ne);

for (i = 0; i < ne; i++)

{

int fd = events[i].data.fd;

if ((events[i].events & EPOLLIN) && (fd == s))

{

memset(buff, 0, BUFF\_SIZE);

n = recvfrom(s, buff, BUFF\_SIZE, 0, NULL, NULL);

if( n == -1)

perror("recvfrom()");

else

printf("Recv %dst message from server:%s\n", times, buff);

}

}

}

err = setsockopt(s, IPPROTO\_IP, IP\_DROP\_MEMBERSHIP, &mreq, sizeof(mreq));

close(efd);

close(s);

return 0;

}